Educational Approach of Refactoring in Facilitating Reverse Engineering

Mazni Omar[1], Sharifah Lailee Syed-Abdullah[2]
Faculty of Computer Sciences and Mathematics
Universiti Teknologi MARA
02600, Arau, Perlis. Malaysia

Kamaruzaman Jusoff[3]
TropAIR, Faculty of Forestry
Universiti Putra Malaysia
43400, Serdang, Selangor. Malaysia

Azman Yasin[4], Haslina Mohd[5]
UUM College of Arts and Sciences
Universiti Utara Malaysia
06100, UUM Sintok, Kedah. Malaysia
[4]yazman@uum.edu.my, [5]haslina@uum.edu.my

Abstract—**Refactoring improves software codes and design. This activity often neglected by software developers because they need time to decide tactically where and when to refactor codes. Although the concepts theoretically instilled in the developer's mind, this activity is not easy to apply and visualize. This situation became more problematic when deals with inexperienced developers. Therefore, there is a need to develop an educational approach to comprehend refactoring activity. This activity was applied through reverse engineering tasks. The software engineering (SE) teams were required to apply reverse engineering activity in order to check the consistency between codes and design. The teams were encouraged to apply Model-View-Controller (MVC) pattern architecture in order to facilitate the activities. Findings revealed that Extreme Programming (XP) teams managed to complete reverse engineering tasks earlier than Formal teams. This study found that the approach is important to increase understanding of refactoring activities in reverse engineering process. This approach will be furthered applied for others SE teams to gain more insight and perceptions towards improving SE course.**

Keywords-refactoring; reverse engineering; software engineering (SE); XP; MVC

I. INTRODUCTION

Refactoring is a process of changing and restructuring software code without changing the system functionality [1],[2],[3] . This practice can promote reusability and easier understanding of the written code [4]. In addition, refactoring can improve quality of code by reducing code complexity [5]. Fowler et al. [2] suggested that code refactoring is based on human intuition because developers can refactor codes that were found smelly in the program. Therefore, Tong [6] listed several smelly code characteristics which includes duplicate code, type code and too many comments. There are several techniques to refactor code such as moving method and extraction of class introduced by Fowler et al. [2]. However, this practice is not easy to apply and visualize, even though, the concepts theoretically instilled in developers' mind.

Mens and Tourwe [3] suggested several steps in conducting refactoring practice because refactoring can assists assists in maintaining consistency between software documentation and refactored codes. Nonetheless, developers still needed to decide tactically where and when to refactor codes. Therefore, several techniques were introduced such as semi-automated approach [7], object-oriented metrics [8], and automated static analysis tools [9] to detect smelly codes. However, using tools does not guarantee refactoring can be applied effectively as reported by Murphy-Hill and Black [10] that many developers failed to use refactoring tool. This is due to limited capability of the tools to refactor codes frequently and maintains healthy codes.

Refactoring practice is one of the core XP practices. Moser et al. [4] conducted a case study to analyze the impact of refactoring on object-oriented (O-O) classes reusability and found that refactoring was able to improve the code reusability. However, Alshayeb [11] reported that refactoring does not necessarily can improve software quality. In order to improve understanding on refactoring, Carlson [12] introduced a refactoring approach that requires writing test cases and refactor these cases. This activity increased students' understanding in keeping healthy codes without degrading the structure. Based on the above research, technical skills through formal effective training and monitoring are required to comprehend this practice.

Reverse engineering was introduced to software engineering (SE) teams to assist consistency between codes and design. Reverse engineering is a process of analyzing and extracting source code program in order to understand system functionality and representation [13, 14]. By doing this activity, organization can maintain and document their software more easily. Due to rapid and continuous feedback in software industry, exploiting reverse engineering in software development activities is crucial. However, continuous

refactoring is required in reverse engineering in order to ensure consistency of the software artifacts [15]. This highlights the important of refactoring in facilitating reverse engineering process.

The objective of this paper is to develop an educational approach of refactoring practice in reverse engineering tasks.

II. MATERIALS AND METHODS

A. Participants

The subjects in this study were third year undergraduate Information Technology students majoring in software engineering. Sixty-six students made up the sample, forming 16 teams. The experiment started in December 2008 and lasted until the end of March 2009.

The teams were randomly assigned to several different projects and were also randomly assigned to experimental XP and control Formal groups. The teams were required to develop a one-year SE project based on clients' needs. There were a minimum of two groups for each project. Each group was required to develop a web-based application using Java Servlets Pages (JSP).

B. The Approach

During the experiment, the XP teams need to produce only Entity Relationship (ER) Diagram and use case mapping with interface design as design documents. Even though XP advocates less design documents, it does not mean designless approach. In contrast, the Formal teams need to generate massive software design documentation (SDD), which consists of Unified Modeling Language (UML) models as their main artifacts in software development phases.

Realizing the importance of reverse engineering in SE, all teams were required to reverse engineer their software codes to check design consistency. This was done when the teams have completed their design phase. The reverse engineering process was conducted using Netbeans Integrated Development Editor (IDE) tool, which is an open source tool.

Smelly codes were detected when several teams were unable to reverse engineer their codes due to poor design. These happened because the teams failed to structure the codes into appropriate design layer, thus mixing up the interfaces, domain logics and database layer into one file. Fig. 1 shows a login.jsp file containing interface coding and SQL database statement. Failure to structure design layer lead to difficulty in understanding and maintaining the codes.

Although some teams successfully reversed the codes using the tool, the teams realized that their codes still contained unused classes, variables and methods. Therefore, to assist further the reverse engineering process, Model-View-Controller (MVC) was recommended. MVC is a pattern architecture that has three different layers, which are the model, the view and the controller [16]. All the teams were required to structure their codes according to the design layer. By using MVC, codes can be separated into different layers; Business (Fig. 2), Data (Fig. 3) and Controller (Fig. 4). In MVC pattern, interface layer consists of HTML and JSP codes. This layer will interact with controller layer, which consist of servlet file. By using Netbeans IDE, all the codes were reversed to UML class diagrams as shown in Fig. 5, and in general, the architecture of software using MVC pattern is shown in Fig. 6.

```
          …….
<%
//connect to database
Statement statement;
Connection connection;

String userName = "root";
String password = "123456";
String url = "jdbc:mysql://localhost:3306/crs";

Class.forName("com.mysql.jdbc.Driver");
System.out.println("Driver loaded");
connection = DriverManager.getConnection(url, userName,
password);
System.out.println("Database connected");

statement = connection.createStatement();

//get input from user
String txtuserName = request.getParameter("userName");
String txtpassword = request.getParameter("password");

//SQL statement
String query = "INSERT INTO logindb VALUES ('" +
txtuserName + "','" + txtpassword + "')";
statement.executeUpdate(query);
%>
           ……
```

Fig. 1 Example of mixed codes (Login.jsp)

```
…..
package CRS.Business;

public class UserLogin {
    private String userName;
    private String password;

    // constructor
    public UserLogin() {
    }

    //constructor with arguments
    public UserLogin(String newUsr, String newPwd)
    {
        userName = newUsr;
        password= newPwd;
    }

    public String getUserName()
    {
        return userName;
    } …..
```

```
@purpose to set and
get user login
information
```

Fig. 2 Example of codes in business layer (UserLogin.java)

```
…..
package CRS.Data;

import CRS.Business.*;
import java.sql.*;
import java.io.*;

public class LoginDb {
    /* Method to match user name and pasword entered by user
with database */
    public static synchronized boolean isMatch(Connection
conn, UserLogin lo)throws SQLException
    {
        boolean status=false;

        String query = "select * from logindb";
        Statement stat = conn.createStatement();
        ResultSet res = stat.executeQuery(query);

        while (res.next())
        {
            if
(res.getString(1).equalsIgnoreCase(lo.getUserName()))&&
(res.getString(2).equalsIgnoreCase(lo.getPassword())))
                status = true;
        }
        stat.close();
        return status;
    }
}…..
```

Fig. 3 Example of codes in data layer (LoginDb.java)

```
…..
package CRS.Controller;

import CRS.Data.LoginDb;
import CRS.Business.UserLogin;
import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginServlet extends HttpServlet
{
    // declare connection
    private Connection conn;
…..
// get information
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException,
ServletException
    {
        String usr = request.getParameter("userName");
        String pwd = request.getParameter("password");
        UserLogin lo = new UserLogin(usr, pwd);

        HttpSession session = request.getSession();
        session.setAttribute("lo", lo);
}…..
```

Fig. 4 Example of codes in controller layer (LoginServlet.java)

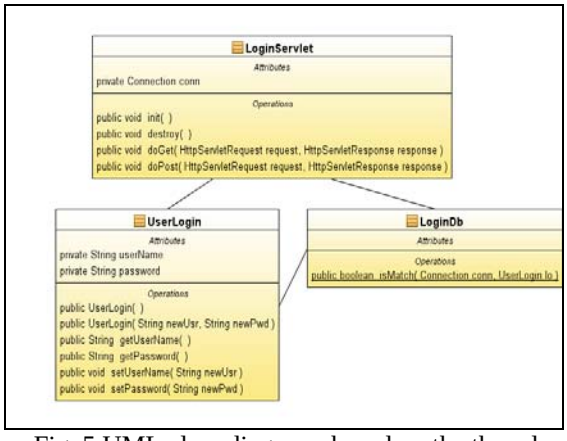

Fig. 5 UML class diagrams based on the three layers (business, data, and controller)

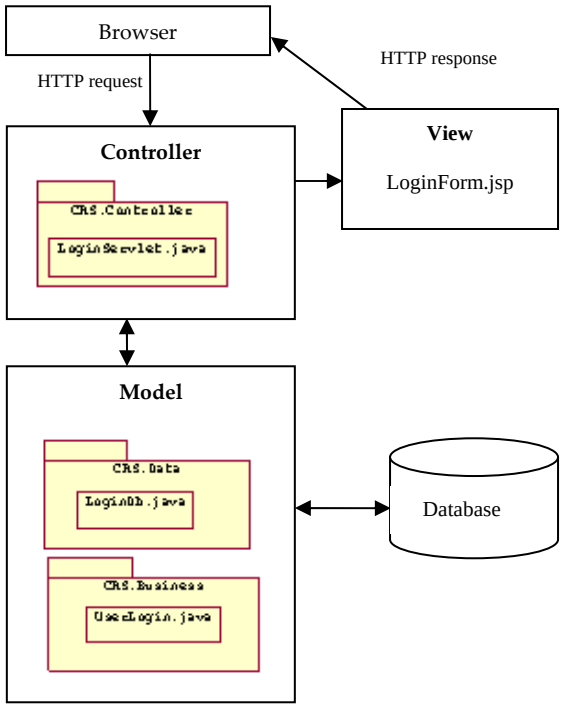

Fig. 6 MVC pattern (adapted from [16])

By insisting that the teams refactored according to these packages, consistency between codes and design document was uphold. This decision promoted higher cohesion and lower coupling codes resulting in improved code design. A general approach of reverse engineering through refactoring is shown in Fig. 7.

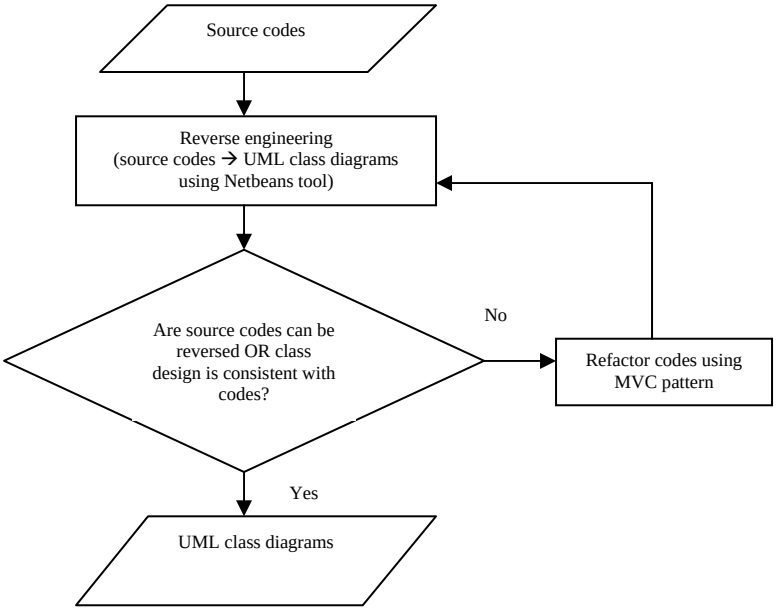

Fig. 7 A general approach of reverse engineering through refactoring

III. RESULTS AND DISCUSSION

In the early part of the experiment, both XP and Formal teams complained that the reverse engineering tasks were troublesome. But, finally all the teams managed to do reverse engineering when the teams refactored the codes using MVC architecture as advised. It was observed that XP teams managed to complete the reverse engineer task earlier than Formal teams. This is because there were fewer documents to refer to when reversing the procedure. Although the design documents act as a blueprint for code development, in XP, design is considered only as an early manifestation of ideas, whereas the coding process allows the developers to realize their idea in a more concrete way [17]. The simple design documents in XP allow the teams to be more flexible when conducting reverse engineering tasks because the existence of only ER diagram and use case mapping with interface design allowed the teams to identify immediately the essential codes. In addition, knowledge sharing between team members through pair programming practice also add up to the simplicity of doing reverse engineering activity. Coding standard, which was fostered by both XP and Formal teams help the team to understand the code easily. This has build up the teams' confidence to refactor these codes.

In comparison, the Formal teams were tied up with the design documentations that they have developed. Table 1 shows a comparison of design documents between XP and Formal teams. Too many design documents must be mapped out during the reverse engineering activities. The teams must prepare a comprehensive documentation consisted of system packages diagrams, class diagrams and interaction diagrams. Furthermore, each classes, methods and attributes identified must describe their purposes and responsibilities. The teams also have to think up-front the algorithms used in the system. Although describing the purposes and responsibilities tasks is good and useful in order to facilitate future documentation references, it dragged the teams towards ineffective solution due to the limited time and budget.

As a result, the Formal teams reverse engineered by only updating to class diagram and describing each class, methods and attributes, but disregarded other design documents. In addition, the teams also failed to update complex algorithm in the interaction diagrams because they discovered this document is unnecessary as reference in reverse engineering activities. This demonstrated that not all documents are necessary to relate code to design activities.

These empirical findings proved that heavy documentation slow down the reverse engineering activities and often SE teams failed to maintain the software documentation. The evidences support earlier studies by Lethbridge et al. [18].

Based on the experiment conducted, the teams realized and appreciated the important of reverse engineering and refactoring activities, because:

- The code is easier to maintain when dealing with requirements uncertainty from clients.
- More codes can be reused and therefore speed up the development process.
- Segregation of programming tasks can easily be achieved among the teams. This allows the effective testing and integrating of codes.
- A design pattern such as MVC is very useful when dealing with web design architecture.

TABLE I
A COMPARISON OF DESIGN DOCUMENTS BETWEEN XP AND FORMAL TEAMS

XP	Formal
1. ER Diagram	1. System architectures a) System packages diagrams b) Class diagrams c) Interaction diagrams (sequence and collaboration diagram)
2. Use cases mapping with interface design	2. Description of each classes, methods, and attributes 3. Algorithm for each methods

IV. CONCLUSION

This study provides an approach to facilitate refactoring practice amongst inexperienced SE teams. Reverse engineering is enhanced through the use of refactoring and by understanding design pattern architecture such as MVC offers the teams a tool to appreciate refactoring activities better. This tool assisted the teams to envisage the structure of the codes. During the experiment, XP teams managed to complete reverse engineering tasks earlier compared to Formal teams. This revealed that XP practices such as refactoring, simple design, pair programming and coding standards are vital in speeding up the reverse engineering activities. Therefore, this study found that this approach is important to increase understanding of refactoring in improving reverse engineering activities. This approach will be furthered applied for others SE teams to gain more insight and perceptions towards improving the SE course.

ACKNOWLEDGMENT

The authors would like to express our appreciation to Mr Zhamri Che Ani and Mr Mohd Zabidin Husin, lecturers, from Universiti Utara Malaysia, who has given their support towards the completion of this research. In addition, we would like to thanks to all clients, supervisors and students in this study.

REFERENCES

[1] K. Beck, *Extreme Programming Explained: Embrace Change*. USA: Addison-Wesley, 2000.
[2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. USA: Addison-Wesley, 1999.
[3] T. Mens and T. Tourwe, "A Survey of Software Refactoring," *IEEE Transactions on Software Engineering*, vol. 30, pp. 126-139, 2004.
[4] R. Moser, A. Sillitti, P. Abrahamsson, and G. Succi, "Does Refactoring Improves Reusability?," in *Lecture Notes in Computer Science(LNCS)*, M. Morisio, Ed. Berlin Heidelberg: Springer-Verlag, 2006, pp. 287-297.
[5] K. Stroggylos and D. Spinellis, "Refactoring-Does it improve software quality?," presented at 5th International Workshop on Software Quality, Minneapolis, 2007.
[6] K. L. Tong, "Essential Skills for Agile Development," Macau Productivity & Tech, 2004.
[7] T. Mens and T. Tourwe, "Identifying Refactoring Opportunities Using Logic Meta Programming," presented at 7th European Conference on Software Maintenance and Reengineering, Benevento, 2003.
[8] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics Based Refactoring," presented at 5th European Conference on Software Maintenance and Reengineering, Lisbon, 2001.
[9] F. Wedyan, D. Alrmuny, and J. M. Bieman, "The Effectiveness of Automated Analysis Tools for Fault Detection and Refactoring Prediction," presented at 2nd International Conference on Software Testing Verification and Validation, Denver, 2009.
[10] E. Murphy-Hill and A. P. Black, "Refactoring Tools: Fitness for Purpose," *IEEE Software*, vol. 25, pp. 38-44, 2008.
[11] M. Alshayeb, "Empirical investigation of refactoring effect on software quality," *Information and Software Technology*, vol. 51, pp. 1319-1326, 2009.
[12] B. Carlson, "An Agile Classroom Experience: Teaching TDD and Refactoring," presented at Agile Conference 2008, Toronto, 2008.
[13] V. C. Garcia, D. Lucredio, and A. F. d. Prado, "Towards an effective approach for Reverse Engineering," presented at 11th Working Conference on Reverse Engineering (WCRE'04), Netherlands, 2004.
[14] I. Sommerville, *Software Engineering 8th Edition*. USA: Addison-Wesley, 2007.
[15] G. Canfora and M. D. Penta, "New Frontiers of Reverse Engineering," presented at Future of Software Engineering (FOSE'07), Minneapolis, 2007.
[16] J. Murach and A. Steelman, *Murach's Java Servlets and JSP*. USA: Mike Murach & Associates, Inc, 2008.
[17] S.-A. Sharifah-Lailee and O. Mazni, "Extreme Programming and Its Positive Affect on Software Engineering Teams" presented at International Conference on Computing and Informatics (ICOCI09), Kuala Lumpur, 2009.
[18] T. C. Lethbridge, J. Singer, and A. Forward, "How Software Engineers Use Documentation: The State of the Practices," *IEEE Software*, pp. 35-39, 2003.

AUTHORS PROFILE

Mazni Omar (Corresponding author) is a lecturer at the College of Arts and Sciences, Universiti Utara Malaysia. Currently, she is pursuing her PhD at the Faculty of Computer Science and Mathematics, Universiti Teknologi MARA, 02600, Arau, Perlis. Malaysia (e-mail: mazni@isiswa.uitm.edu.my). Her main research focuses are agile methods in software engineering, empirical software engineering and pattern recognition.

Sharifah Lailee Syed-Abdullah is an Associate Professor at the Faculty of Computer Science and Mathematics, Universiti Teknologi MARA, 02600, Arau, Perlis. Malaysia (e-mail: shlailee@perlis.uitm.edu.my). Her main research interests are agile methods in software engineering, empirical software engineering and pattern recognition.

Kamaruzaman Jusoff is a senior Professor at the Faculty of Forestry, Universiti Putra Malaysia, 43400 UPM, Serdang, Selangor, Malaysia. (e-mail: kamaruz@putra.upm.edu.my). He is a prolific academician, specialised in remote sensing field. He has contributing numerous writings in interdisciplinary fields in high impact and citation index (CI) journals.

Azman Yasin is a senior lecturer at the College of Arts and Sciences, Applied Science Division, Universiti Utara Malaysia, 06010 UUM Sintok, Kedah. Malaysia (e-mail: yazman@uum.edu.my). His research interest includes software engineering education, information retrieval specifically scheduling and timetabling using artificial intelligence techniques.

Haslina Mohd is a senior lecturer at the College of Arts and Sciences, Applied Science Division, Universiti Utara Malaysia, 06010 UUM Sintok, Kedah. Malaysia (e-mail: haslina@uum.edu.my). Her research areas are within software engineering education and health informatics.